REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	24 Jul 01	Final Report 24 Jan 01 - 24 Jul 01

4. TITLE AND SUBTITLE

Software Agent Technology for Large Scale, Real-Time Logistics Decision Support

5. FUNDING NUMBERS

Contract # DAAD17-01-C-0023

6. AUTHOR(S)

Dr. C. Allen Butler, Mr. James T. Eanes

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Daniel H. Wagner Associates, Inc.
2 Eaton Street, Suite 500
Hampton, VA 23669

8. PERFORMING ORGANIZATION REPORT NUMBER

Case 7780

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
Aberdeen Proving Ground
APG, MD 21005-5001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

20010731 066

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT (see Section 5.3b of this solicitation)

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Report developed under SBIR contract for topic A00-114. Software agents are computer programs that perform specific operations for users based on those users' wishes or needs. The benefit of an agent is that it can be programmed to provide precisely the information needed by that user, at the appropriate time and place for that information to be useful. Agents can save users enormous amounts of time by avoiding the need for developing complicated searches and extract scripts every time new information is required. This report details the results of a Phase I SBIR research project for the US Army on the concept of a Logistics AgentWizard™ (LAW), a program that can automatically create agents for users without the need for detailed programming or significant computer knowledge on the user's part. In Phase I we designed the architecture of the LAW and developed a prototype user interface to demonstrate the concepts of user-driven agent creation. We also developed Java Bean agent components that can be combined within the LAW to create agents capable of providing automated assistance to current and future logistics operations.

14. SUBJECT TERMS

SBIR Report, Logistics, Software Agents, Agent Wizard, DAML, XML

15. NUMBER OF PAGES

22

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	unclassified	unlimited

REF E

Case: 7780

Software Agent Technology for Large Scale, Real-Time Logistics Decision Support

Phase I Final Report
SBIR Topic A00-114

Contract Number: DAAD17-01-C-0023

July 24, 2001

Prepared For:
US Army Research Laboratory
Aberdeen Proving Ground, MD

Prepared by:
Daniel H. Wagner Associates, Inc.
2 Eaton Street, Suite 500
Hampton, VA 23669

Dr. C. Allen Butler
Mr. James T. Eanes

Executive Summary

Software agents are computer programs that perform specific operations for users based on those users' wishes or needs. The benefit of an agent is that it can be programmed to provide precisely the information needed by that user, at the appropriate time and place for that information to be useful. Agents can save users enormous amounts of time by avoiding the need for developing complicated searches and extract scripts every time new information is required. In the context of military mission planning, an agent being developed by Daniel H. Wagner Associates, Inc. called "METPLAN" will provide a cost effective means for developers of mission planning decision aids to include environmental information. This report details the results of a Phase I SBIR research project for the US Army on the concept of a Logistics AgentWizard™ (LAW), a program that can automatically create agents for users without the need for detailed programming or significant computer knowledge on the user's part. In Phase I we designed the architecture of the LAW and developed a prototype user interface to demonstrate the concepts of user-driven agent creation. We also developed Java Bean agent components that can be combined within the LAW to create agents capable of providing automated assistance to current and future logistics operations.

Table of Contents

List of Figures

1. Overview of Phase I

1.1 Project Goals

The goal of this SBIR project is to provide a means by which end-users can easily create software agents to assist logistics operators as they perform their daily tasks. We accomplish this goal by developing a tool for creating software agents and by providing a framework for future work on developing software agents in the logistics information space.

We have designed a tool, which we call the Logistics AgentWizard™ (LAW), for creating and controlling software agents. Agents, in general, are persistent software programs capable of providing specialized, automated, user-oriented services over time. The agents created by the LAW will be responsible for locating, retrieving, and processing data from information resources on heterogeneous networked systems according to a set of rules and preferences created by the user.

The prototype system we have developed in Phase I contains two types of graphical user interface (GUI): one to allow the user to graphically combine Java Bean agent components into working software agents; and another to allow the user to create an agent by defining the rules and preferences for the agent's behavior within the given knowledge domain. The first type of GUI, called the Agent Builder, provides a toolbox, from which the user selects existing Java Bean agent components, and a work area, in which the user combines these components into an agent and defines the agent's behavior through graphical manipulation of each component's properties and events. The second interface is a set of Requirements Wizard screens designed toward specific operations within the logistics domain. For example, there is a Requirements Wizard for creating agents to track packages handled by commercial shipping companies (e.g. FedEx, UPS). Another Requirements Wizard is designed toward creating agents to retrieve and display weather data. This type of interface is designed to require very little computer familiarity on the part of the user, whereas the Agent Builder requires the user to be comfortable with graphical manipulation of Java Beans.

1.2 Phase I Accomplishments

In Phase I, we have identified several logistics operations that would benefit from software agent assistance, including automated tracking of shipments and retrieval of weather data. We have designed the LAW architecture, which consists of the following components: 1) specialized Requirements Wizards for easy creation of agents capable of supporting well-defined logistics operations; 2) an Agent Builder for graphical manipulation of agent components; and 3) an Agent Monitor for controlling agents once they've been deployed. We have developed specialized Java Bean agent components capable of being combined into working agents to fulfill specific logistics-oriented requirements. We have developed a prototype of the Agent Builder, which allows manipulation of our Java Bean agent components to create working agents. We have developed prototype Requirements Wizard screens, which demonstrate the steps an end-user would take to define agents within the scope of specific logistics operations. And, finally, we have laid the foundation for future work in Phases II and III by continuing our study of current and future agent and component architectures within the military and commercial sectors.

1.3 Agents in the Information Space

When an agent is created and dispatched by the LAW, that agent attempts to locate, retrieve, and process data within the given information space. In the case of the military, the information space consists of five tiers: direct sensor connections (e.g. wireless and serial data streams), Local and Wide Area Networks (LANs and WANs), the Global Command and Control System (GCCS), the Secure Internet Protocol Routing Network (SIPRNET), and the Internet. Agents responsible for direct sensor communications will be programmed toward those sensors' interfaces (e.g. RS-232, TCP/IP). Agents operating on military LANs/WANs will communicate with each other and with local data sources (e.g. proprietary databases, Intranet web pages) using published application programming interfaces (APIs) and standard network protocols (e.g. TCP/IP, UDP). On GCCS, agents will communicate with segments through their published interfaces and standard GCCS protocols (e.g. OTH-Gold messages and overlays). SIPRNET agents will access web servers that are under military control, and Internet agents will access web servers under the separate control of individual corporate entities. For commercial agents, the information space will consist of all the above, except GCCS and SIPRNET.

The architecture of the information space will determine each agent's ability to automatically assist the end-user in his daily operations by dynamically discovering data sources and interacting autonomously on the network. All agents will be restricted in their behavior according to the data sources and metadata available on the networks on which the agents operate. For example, an agent designed to operate on a network enabled with the Common Object Request Broker Architecture (CORBA) will be able to dynamically determine data source accessibility through the Object Request Broker's (ORB) Interface Repository. This agent has much more dynamic capability than an agent operating on the Internet, on which most web servers contain no metadata descriptions of the web pages they serve. The Internet agent is currently forced to operate based on pre-programmed understanding of web server query structures and web page layouts ("screen-scraping"), whereas the CORBA agent is able to dynamically interact with the network to make decisions based on data availability. As new technologies emerge for deploying metadata on the Internet, such as eXtensible Markup Language (XML) and the DARPA Agent Markup Language (DAML), and on commercial and military Intranets, such as CORBA, the Data Exchange Infrastructure (DEI), and the Joint Battlespace Infosphere (JBI), agents will become more capable of dynamically interacting with the information space. This will reduce the amount of human interaction required for agent processing, allowing them to more effectively meet end-user requirements.

Our approach to agent development also requires metadata descriptions of the network infrastructure and the components from which agents will be created. In Phase I, we have developed several agent components capable of operating on the sensor, LAN/WAN, and Internet tiers of the information space, and we have developed prototype metadata descriptions to assist in the creation of agents from these components. In the creation of our components, we have assumed the presence of no metadata on the network at this time (i.e. none of them are XML-, DAML-, or CORBA-enabled). During Phase II, we plan to enable our components with some level of metadata understanding, as those technologies continue to mature.

2. Phase I Technical Objectives and Specific Accomplishments

2.1. Technical Objectives

The ultimate goal of this research effort has been to demonstrate improved logistics operations through the use of software agents. In Phase I, we have attempted to demonstrate the feasibility of this concept. To accomplish this we set ourselves three primary objectives.

Objective 1. Demonstrate improved operations in selected logistics applications with the use of intelligent agents.

Objective 2. Demonstrate a method for easily creating and controlling software agents.

Objective 3 (Option). Lay the groundwork for future development of a multi-agent system.

The remainder of this section describes the details of our accomplishments in Phase I.

2.2. Logistics Applications

On February 22, 2001, Dr. Butler and Mr. James Eanes traveled to Aberdeen Proving Ground, MD to attend the kickoff meeting for the subject contract. The government representatives attending the meeting were Mr. Tim Hanratty and Mr. John Dumar. Mr. Hanratty provided an overview of previous work that led to their interest in the logistics application. Dr. Butler and Mr. Eanes presented background information and a detailed outline of the proposed tasks to be completed under this contract.

Our approach to identifying the appropriate logistics applications has been three-fold. First, we performed an extensive literature review, obtaining relevant documents primarily over the Internet. Second, we are pursuing commercial application of this technology, specifically to support the Army's desire to move towards a greater level of contractor logistics support. Our primary marketing opportunity in the commercial arena is United Defense, LP. Third, we briefed personnel at Fort Eustis and Fort Lee to extract expertise from the various logistics and digital logistics experts that work at these bases.

Our literature search resulted in the identification of a number of programs that may benefit either directly or indirectly from the application of intelligent software agent technology. These programs/products include: Combat Service Support Control System (CSSCS), Global Combat Support System (GCSS-A), the DARPA Advanced Logistics Project, the Integrated Logistics Analysis Program (ILAP), the Automated Battlebook System (ABS), the Army War Reserve Deployment System (AWRDS), the Failure Analysis and Maintenance Planning System (FAMPS), and the Logistics Integrated Database (LIDB). Clearly, there are a large number of programs related to automation/digitization in the field of Army logistics. The Logistics Agent Wizard will not duplicate nor replace any of these programs. The intelligent application of agent technology will enable the end-user to optimize the performance of the aforementioned systems. Furthermore, the Logistics Agent Wizard will work with existing legacy (even highly stovepiped) systems, as well as the new systems coming down the pipeline. It may also provide the ability to fill in any gaps created by inconsistencies between systems.

We have been involved in ongoing discussions with Mr. Bill Altergott and Ms. Dawn Brown of the Armament Systems Division of United Defense, LP. Under the new Army of Excellence (AOE) concept, some logistics functions related to major armaments are being relegated directly to the contractor that builds/supplies the armament. This concept falls under the umbrella of Contractor Logistics Services. Implementing a Logistics Agent Wizard, designed specifically to support this concept would provide United Defense a competitive advantage as a total system integrator. On May 16, 2001, we briefed our agent technology and discussed potential teaming arrangements with UDLP. Mr. Altergott indicated that they might provide engineering support, direct financial support or a combination of both. Based on the outcome of that meeting, Mr. Altergott provided a letter of support for the Phase II proposal.

On May 7, we traveled to Fort Eustis, VA at the request of Mr. Martin Walsh of the US Army Aviation Applied Technology Directorate (AATD). We briefed several members of Mr. Walsh's staff as well as Captain Jim Barnett and his staff, who support Col. Bonham, chairman of the Aviation Maintenance and Support–Process Action Team (AMASS-PAT). Their response was enthusiastic and encouraging and Mr. Walsh is putting together a list of contacts that he believes will provide us with a number of possibilities for applying this technology.

Based on our discussions at Fort Eustis, we have several ideas for improving logistics operations using this technology. For example, suppose an Operational Order has been received to deploy an Aviation Task Force into Kosovo. Due to geographical restrictions, NATO has established a limitation on the size of that force. The Task Force Maintenance Officer is tasked by his Commander to build the appropriate maintenance support package for thirty helicopters, but to construct the package smaller than usual. Location and transportation will limit the size and mobility of the support package. He will be restricted to a single container of spare parts, when normal logistics planning for the contingent of thirty helicopters could dictate three containers of parts. With the LAW as an enabling agent technology, the officer could have several agents at his disposal to assist him in quickly and efficiently performing his duties. For example, one agent calculates a priority ordered list of parts for the specific mix of helicopter types in the deployment using standard logistics predictive maintenance tables. A second agent queries the historical database to determine the parts that most commonly fail for these particular helicopters. A third agent queries a system such as the Failure Analysis and Maintenance Planning System (FAMPS), which uses embedded diagnostics to help predict failures. The amalgam of information provided by these three techniques is then presented to the maintenance officer in a standard format, allowing the officer to make an informed decision on the appropriate supplies. Note that the functionality will still be present even if one or two of the agents are unable to perform their task due to the absence of, or the inability to communicate with the corresponding system/database. In other words, as long as there is at least one data source available, the maintenance officer will be provided with supporting information for making his decisions.

As a second example, consider the following scenario after base camp has been established. The Production Control Clerk in the Aviation Intermediate Maintenance Unit (AVIM) has received work orders from customer units and processes them into the Standard Army Maintenance System (SAMS). The Maintenance Officer would like to know if the repair parts are available to complete the work orders. The LAW would provide the officer with an agent to assist in this task. The agent might first determine what version of the SAMS is being operated at the AVIM. Based on the specific version, the agent builds the appropriate query to determine whether or not the required parts are on hand in SAMS Shop Stock. If the parts are unavailable, a second agent is activated. This agent will determine, in a hierarchical fashion, which systems or database it must query (e.g., Standard Army Retail Supply System (SARSS) in the Supply Support Activity) in order to find the location of the parts that can be most rapidly requisitioned. Upon confirmation by the officer, the requisition order will be automatically generated and the appropriate information will be passed to SARSS. Once the Estimated Ship Date (ESD) has been estimated for the availability of the parts, another agent will query the Standard Installation/Division Personnel System (SIDPERS) in the Company Orderly room and generate a list of repair technicians capable of performing this repair, indicating their location and duties at the time of the parts availability. This will allow the maintenance officer to make the correct decision, and properly plan for, the dismantling of the aircraft.

Finally, suppose that during the initial phase of the operation the connectivity of both the local area network and the wide area network is unreliable. With the classic style of computer program, the operator would have to sit at the laptop and regularly test to see whether or not his requests were making it through. Using software agent technology, the agent would monitor the network continually and would send the queries when it was able. This autonomous behavior is one of the key strengths of software agent technology. Note that this capability overcomes not only the problems associated with physical imperfections of the network, but also procedural limitations. For example, if the bandwidth is highly limited, as is often the case during the initial phase of any operation in a remote locale, a common tactical procedure is to restrict the use of the Internet connectivity for the commander during significant portions of the day. Through the use of intelligent software agents, the bandwidth can be shared and thus used more efficiently, while guaranteeing that the commander's requests for information are not impeded in the slightest.

On June 12, Dr. Butler and Mr. Eanes traveled to Fort Lee, VA and met with LTC Thet-Shay Nyunt, Project Director Anticipatory Logistics Experiments, Combined Arms Support Command. We provided a briefing on the Logistics Agent Wizard. LTC Nyunt believes that this technology has great potential and is attempting to arrange for us to brief the Program Manager for the Integrated Logistics Analysis Program (ILAP). We also briefed LTC Green, the Directorate of Combat Developments for Petroleum, Oils and Lubricants (POL). She also described a number of areas where she believes the LAW could provide assistance in her duties. For example, during the planning of POL supplies, the current technique is to use "back of the envelope" calculations. By looking at the desired OPTEMPO, topography, and equipment, an agent could automatically calculate a far more accurate estimate of the required POL supplies and provide this estimate to the logistics planner.

We have also been in touch with Dr. Simmonds of the Army Logistics Management College (ALMC), and Mr. Brian Wood of the Information Systems Directorate (ISD) of the Combined Arms Support Command (CASCOM), both of whom are located at Fort Lee, VA. Dr. Simmonds expressed interest in the idea of an intelligent intranet-based agent for the ALMC. The primary goal of such an agent would be to search the intranet to find the data necessary to better manage the classes and facilities of the College.

2.3. End-User Interface Requirements

Our primary goal in developing the LAW is to enable personnel to easily and efficiently create software agents from reusable components to assist logisticians in their daily tasks. It has been our intention from the start of the project to design an interface for the LAW that is easy to use regardless of the user's level of technical expertise. This intention was mainly driven by the understanding that logistics personnel do not necessarily have experience writing software. For this reason, we initially designed two types of interface to the LAW: 1) Requirements Wizards, which break down logistics tasks into their component parts and step the user through the creation of agents to meet those tasks; and 2) the Agent Builder, which allows users with more technical expertise to fine-tune the agents they create by directly manipulating their properties. We chose this design to allow the maximum number of end-users to be able to create agents, while providing more skilled users with the power of directly controlling agent behavior.

During Phase I, we discovered another reason for designing the LAW interface in this way. From our discussions with logistics personnel at Ft. Eustis and Ft. Lee, it has become apparent that the Army's requirements for control of agents and their behavior will dictate that the person creating the agent will probably not be the person who directly benefits from the agent's assistance. An agent's life cycle may start with a Lieutenant Colonel, who uses the LAW to create agents to be placed in a central agent repository or to be distributed to his subordinates over the network. Depending on the command, those subordinates may then be allowed to clone/modify the existing agents for their own purposes. In any case, the agents would then perform their duties on the network, interacting with the end-user, either directly or through the chain of command, improving the end-user's ability to retrieve and process data into useful information. This type of command-driven requirement for control of agent creation is clearly mimicked in the commercial world, as well, and requires that the LAW interface make no assumptions concerning the technical expertise of the personnel creating agents.

The next three sections describe the LAW prototype that we have developed, from the two types of interfaces to the underlying agent component toolbox and the XML-based process that combines those components into working agents.

2.4. Prototype Requirements Wizards for the LAW

In order to allow any logistics end-user to create software agents, we must break down the tasks involved in the agent creation process into specific requirements, and then step the user through those requirements in a simple, straightforward fashion. We have done this in a very limited way with two tasks that logistics personnel might perform in their daily operations: tracking shipments and gathering weather information.

Figure 1 shows the initial Requirements Wizard interface. This first screen allows the user to choose from the possible logistics application domains currently known to the LAW.

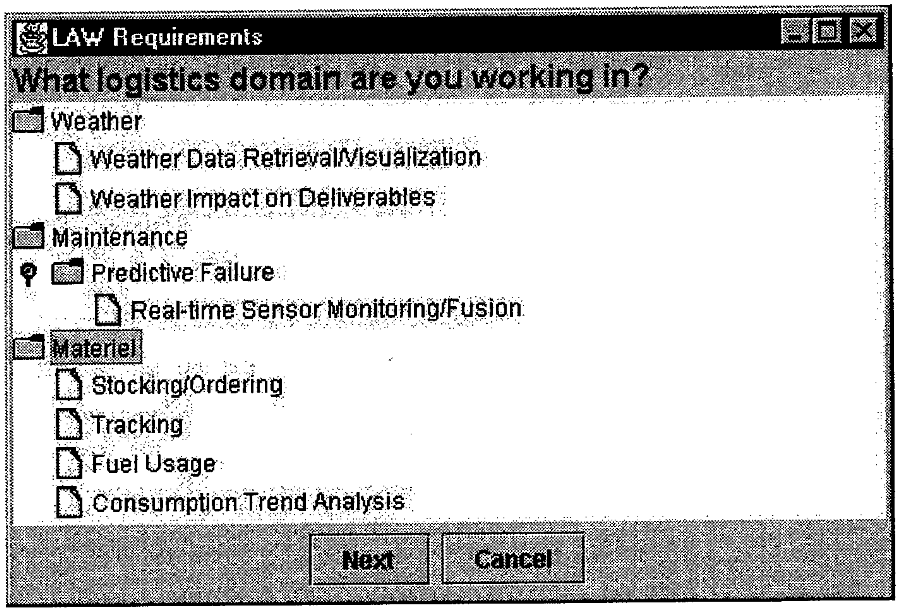

Figure 1. LAW Requirements Wizard Interface

When the user chooses the domain for which they want to create an agent, the appropriate domain-specific Requirements Wizard is invoked. For example, choosing the Tracking domain in the initial screen and clicking "Next" brings up the first screen shown in Figure 2, which asks the user to choose which carrier is shipping the package of interest. Choosing FedEx in that screen and clicking "Next" brings up the second screen shown in Figure 2. Here the user enters the details required to define an agent's behavior: the FedEx tracking number, whether they want to be alerted by email, their email address, whether or not to pop up an alert window when the package arrives at its destination, and how often to check the FedEx server for updated information. With these details in place, the LAW would be able to dispatch an agent with the desired behavior (requiring that the Agent Component Toolbox contain the components necessary to carry out this operation, which will be discussed in a later section).

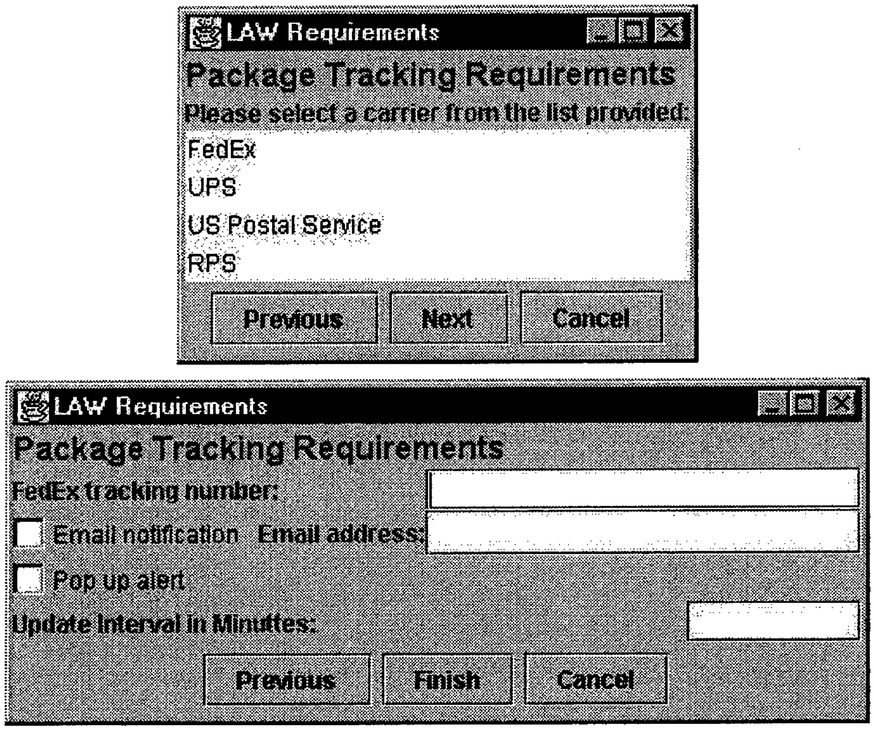

Figure 2. Tracking Requirements Wizard

Another domain of interest to the logistics planner is the weather. Figure 3 shows the Requirements Wizard designed for retrieval of weather information. The first screen asks for the specific meteorological/oceanographic (METOC) information required by the end-user. The next screen allows the user to define the area of interest (AOI). The next screen defines the time for which the METOC data should be valid and how often to update the data. In the next screen, the user chooses which METOC parameters he wants to locate and retrieve. The next screen shows which data sources on the network are currently capable of providing the required data. The last screen allows the user to define what operations should be performed on the data once it is retrieved. Given the preferences collected from the user in these screens, the LAW would be able to create an agent to automatically retrieve and process the required weather data.

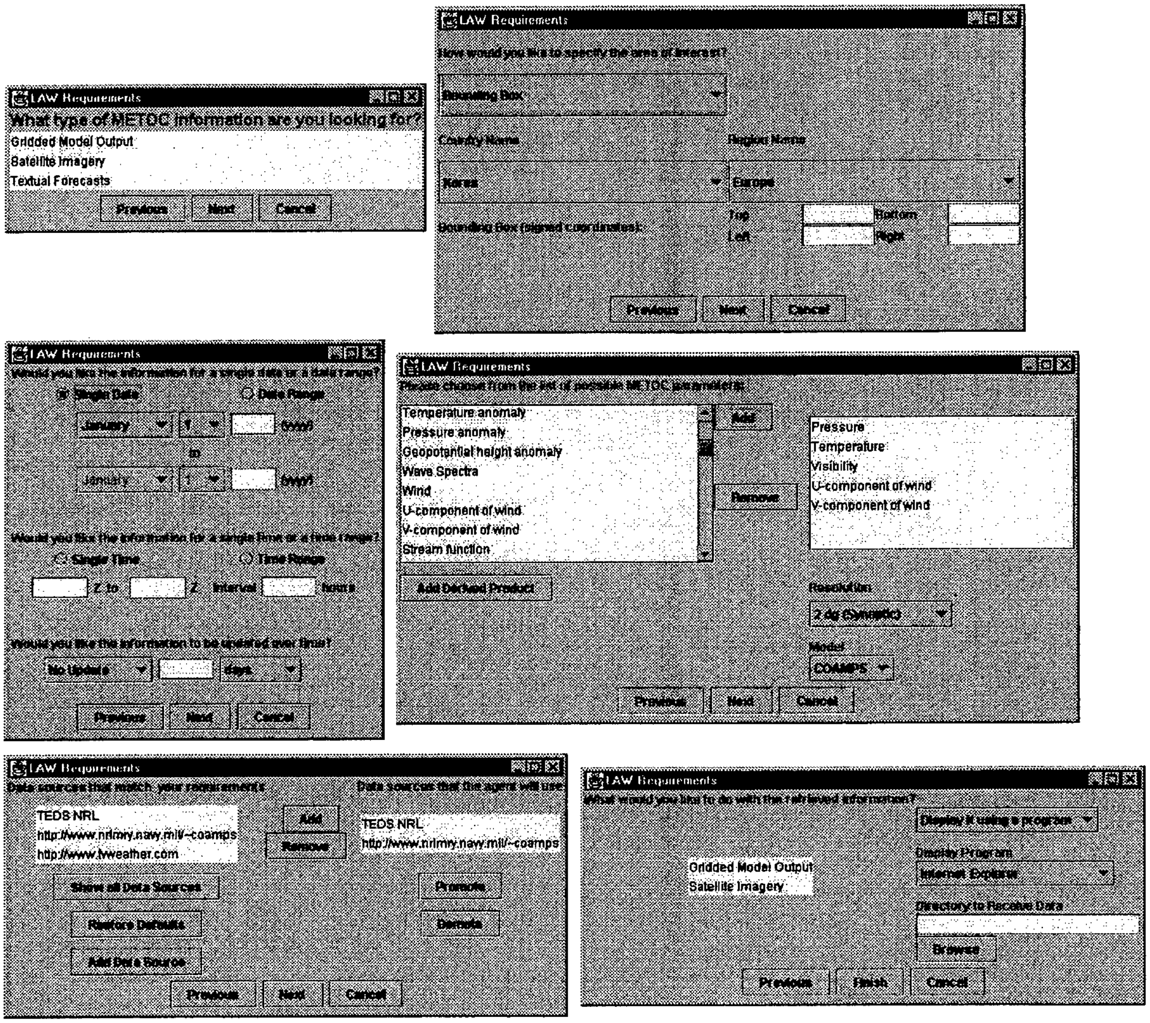

Figure 3. Weather Retrieval Requirements Wizard

Each Requirements Wizard developed for a specific logistics application requires programming. A programmer developed each of the screens shown above for the specific purpose of being displayed within its respective Requirements Wizard. This represents the tradeoff between ease of development and ease of use. In order for end-users with very little technical expertise to be able to easily develop agents to assist their daily operations, time and money must be put into the development of this type of step-by-step interface. The alternative approach, represented by the Agent Builder, is to develop a generic interface that allows for the creation of agents within any application domain, but requires more technical expertise on the part of the end-user.

2.5. Prototype Agent Builder for the LAW

Figure 4 shows the prototype Agent Builder interface. This interface is a simple Java Bean manipulation tool that allows technically inclined end-users to combine Java Bean agent components into working agents. The leftmost screen shows the Agent Component Toolbox, which contains all the components currently known to the LAW. The middle screen shows the Agent Builder palette where components are combined into a working agent. In this example, the user has dragged and dropped two components from the toolbox, getWebGIFs and plotToSystemChart, and placed them on the palette. The red line between the components shows that an event or property from the first component is bound to the second component, which means that the second component's behavior is linked to that of the first (e.g. plotToSystemChart won't activate until getWebGIFs fires an event stating that it has retrieved imagery to display). The rightmost screen shows the Properties window, which shows the user the bound properties for the selected component.

As you can see, this type of interface provides the user with a lot of control over the creation of an agent, but it also requires more technical understanding than the Requirements Wizard interface.

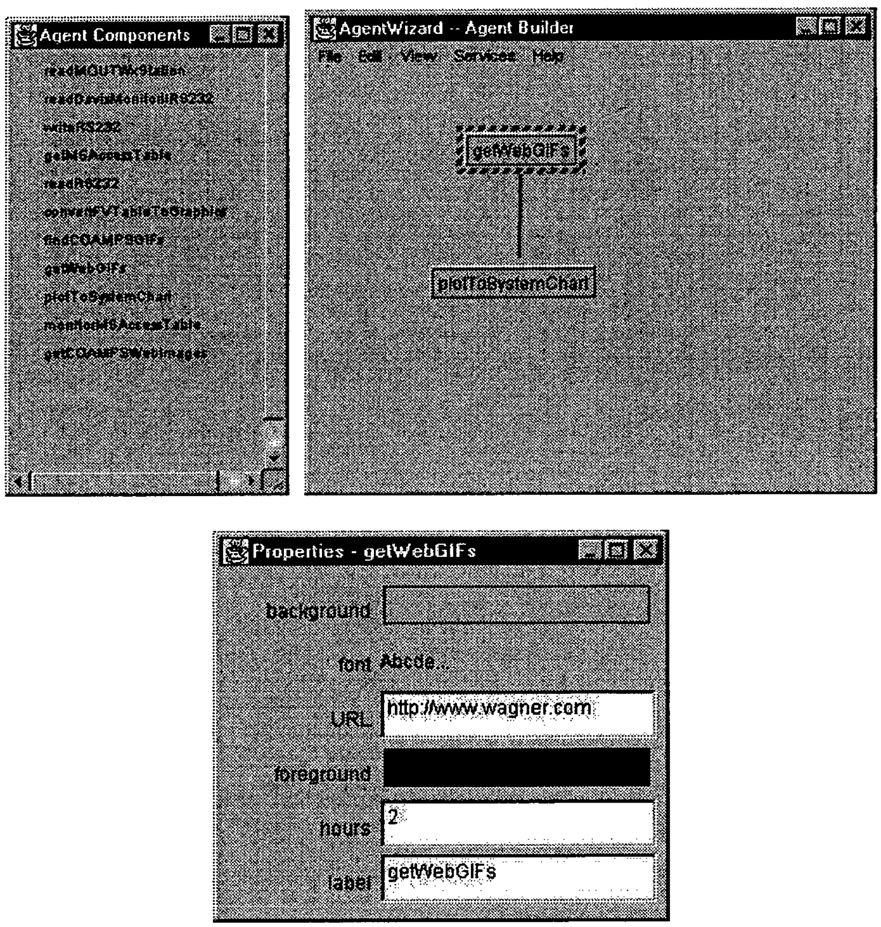

Figure 4. Agent Builder Interface

2.6. The Agent Component Toolbox

Since our primary goal is to allow end-users to create and control agents to assist with their daily operations, we have developed several agent components in Phase I that are capable of performing limited but meaningful tasks in the military information space. These agent components form the basis of the component toolbox from which end-users will create working software agents by means of Requirements Wizards or the Agent Builder. We focused our Phase I component development on general tasks that would be of use in multiple logistics applications. These agent components are capable of operating in the sensor, LAN/WAN, and Internet tiers of the military and commercial information space.

The first component we developed was a generic GIF retriever called getWebGIFs, which is capable of pulling GIF images from the Internet and storing them in a directory. Its bound properties are the file name containing the URLs of the desired images and the directory to place the images when retrieved. This component's usefulness is best illustrated by its use with other components. For example, if a logistician requires knowledge of the natural and urbanized terrain in an area of interest, he can create an agent to locate all SIPRNET web sites containing photographic or hyperspectral images of the area and download those images for separate inspection in his web browser. In this example, the getWebGIFs component would be included in an agent along with a web crawler responsible for locating the URLs of the desired photographs. The web crawler would output a file containing the desired URLs, and then getWebGIFs would retrieve those GIFs to the local machine. By separating the actions of the web crawler from those of the retriever, we increase the speed of the overall retrieval by allowing each to run in its own thread. This way, neither component is directly delayed by the other's data download operations (e.g. once the web crawler has searched a remote site and output the pertinent URLs, it need not wait for getWebGIFs to download those URLs in order to continue searching other sites). Also, this allows any other component to be added to the agent to perform further processing. For example, the user might want to filter the URLs through another component that searches for time-specific information, so that only those images created after a certain date are actually downloaded.

We developed another Internet agent component called getCOAMPSWebImages, which is designed to download images from the Naval Research Lab (NRL) – Monterey's Coupled Ocean-Atmosphere Mesoscale Prediction System (COAMPS) web site. This component is designed to download all available wind streamline products (GIF images) that are valid within a specified number of hours from present time. This component is itself a stand-alone agent, but could be broken into components for locating, retrieving, and then displaying the desired images.

In the sensor interface tier of the information space, we developed several components capable of operating on serial (RS-232) data connections. One component called readRS232 reads ASCII data over a serial connection and outputs that data into a text file. Another called writeRS232 reads in a text file and writes the ASCII data over a serial connection. In order to support a real-world requirement, we developed an agent component capable of reading the binary RS-232 data output by the Davis Monitor II weather station, which is currently in use at the McKenna Manned Operations in Urbanized Terrain (MOUT) site at Ft. Benning, GA. The agent component's output is an XML file containing the weather data from the station, including wind speed and direction, temperature, and pressure. We installed and tested this agent component at the MOUT site, displaying its XML-wrapped output in an HTML file generated by an eXtensible Style Sheet (XSL) document.

In order to demonstrate the concept of agent assistance in the integration of legacy logistics systems, we developed a fully functional agent built from several components that link the displays of two existing systems. The first legacy system was our product called METPLAN, which monitors mission planning activity (missile route changes, etc.) in the Portable Flight Planning Software's (PFPS) map server (FalconView) and seamlessly integrates environmental data into the mission plan. The other legacy system is the GCCS System Chart. In order to integrate these two systems, the agent monitors the graphics displayed by METPLAN onto FalconView in order to display those same graphics on the System Chart. In terms of a real-world scenario, having agents monitor mission planning activity and display that activity on a common tactical picture would provide high-level mission planners with a consolidated view of the battlespace. The three agent components developed to meet this requirements are the following: 1) monitorMSAccessTable, which runs as a separate thread from the agent's main thread in order to constantly monitor changes in the Microsoft Access database table in which METPLAN places graphics for display on FalconView and output those graphics into an ASCII text file; 2) convertFVTableToGraphics, which reads in the graphics text file, converts the given graphics table into a generic graphics description consisting of Lines, Polygons, etc., and outputs those graphics to another text file; and 3) plotToSystemChart, which reads in this graphics text file and displays the graphics on the System Chart.

2.7. The Underlying Technology

Behind both the Requirements Wizard and Agent Builder interfaces is the underlying technology that enables the dynamic combination of Java Bean components into working software agents. During Phase I, we have refined our initial concept of the agent creation process to include metadata documents describing the environment within which the agent is created and deployed. In this section, we will fully describe the underlying technologies involved in this dynamic agent creation process, beginning with the Java Bean architecture and the reasoning behind our choice to use this new component design methodology.

The agent components we have developed are Java Beans, which are reusable Java components that can be manipulated within a GUI and combined dynamically using a process called introspection [9]. Introspection can consist of the standard low-level reflection that is inherent in all Beans, or it can consist of higher-level reflection based on a specific Bean's interface extensions. Each Bean exposes bound properties for manipulation and fires events to be received by the main class' thread or by other Beans. Bound properties are values that can be changed by calls to get- and set- functions. For example, if a Bean contains a property called URL, then it exposes the methods getURL and setURL for reading and writing that property's value. Also, each Bean that is interested in notification of events registers for those events. This event handling is typically used for button and mouse notification in graphical interfaces, but this event service easily extends to include message passing between Beans.

Our choice to develop agent components as Java Beans was based on the need for the components to be cross-platform, atomic, reusable, standard-interface software modules that perform specialized actions on behalf of the user. Currently, Java Beans are most widely used as graphical components for programmers to include in their GUI applications using a Bean-compliant integrated development environment (IDE), such as Sun's Forte or Borland's JBuilder.

Since our agent components are Java Beans, we can combine them automatically without user interaction using introspection, while still being able to graphically display the Beans to the user with standard graphical interface methodology. In order to graphically present agent component Beans to the user, we have extended a demonstration program called the BeanBox, which is included in the Beans Development Kit 1.1 located on Sun's Java web site [10]. The BeanBox has formed the basis of the Agent Builder interface for creation of agents through manipulation of Bean components. As described in the above section, this interface currently allows the user to select components from the Agent Component Toolbox, place them on the Agent Builder's palette, configure their bound properties, and create an agent from the chosen Beans by clicking on File->Make Agent in the Agent Builder window. The created agent is a simple Java class, named by the user, containing a main function that invokes the run methods of the chosen Beans in the chosen order.

We have developed several agent components that form the basis for the Agent Component Toolbox, as described in the previous section. In order for the LAW to dynamically combine existing components to satisfy user requirements, whether those requirements are captured using the Requirements Wizard or the Agent Builder, there must be metadata descriptions of the environment in which the agent is created and will be deployed. In the current architecture, we have metadata files describing each of the following: 1) the network infrastructure (databases, web sites, agents, sensors, etc.); 2) all available software components (JavaBeans, C++ classes, existing agents); 3) and the grammar describing the actions that can be taken by components within the given knowledge domain (download imagery from a URL, query a database, communicate with another agent, etc.). The infrastructure and component descriptions are written in XML documents, and the grammar is in a standard Backus-Naur Form (BNF) text file. These descriptions allow the LAW to dynamically compile software components into useful agents within the constraints of the end-user's knowledge domain and network environment.

The example infrastructure description shown in Figure 5 demonstrates the example of requiring weather data for a specified time from a specified model, namely COAMPS. The data sources associated with COAMPS in the description are the Tactical Environmental Data Server (TEDS) existing on a local machine and the Naval Research Lab (NRL) – Monterey COAMPS web site. The example component description in Figure 6 shows that there is one component (getCOAMPSData) capable of getting data from TEDS. However, this component outputs raw data as ASCII text and imagery as NITF. The listing also shows an agent that outputs names of GIF image files from COAMPS (findCOAMPSGIFs), but not the GIFs, themselves. A third component (getWebGIFs) retrieves GIF files from an HTTP source, but is not linked explicitly to COAMPS. In order for the LAW to effectively respond to the user's request for GIF images from COAMPS, it must combine the getWebGIFs and findCOAMPSGIFs components into one agent that seamlessly outputs the desired imagery.

```xml
<infrastructure location_type="LAN" location ="38.229.53.255">
  <domain name="logistics" model ="E:\Users\James\DataModels\logmodel.dtd">
    <subdomain model="E:\Users\James\DataModels\metoc.dtd>weather</subdomain>
  </domain>
  <local network="10/100T Ethernet">
      <source type="database">
        <name>TEDS</name>
        <alias>Tactical Environmental Data Server</alias>
        <alias outdated="true">NITES</alias>
        <url>medal.va.wagner.com</url>
        <vendor>Informix</vendor>
        <access>SQL</access>
        <access>API</access>
        <access>Metcast</access>
        <data>
          <originator type="model">
            <name>COAMPS</name>
            <alias>Coupled Ocean-Atmosphere Mesoscale Prediction System</alias>
            <location>NRL-Monterey</location>
          </originator>
          <output type="raw data" format="GRIB" units="m/sec">wind
             speed</output>
          <output type="raw data" format="GRIB" units="deg">wind
             direction</output>
          <output type="raw data" format="GRIB" units="K">temperature</output>
          <output type="imagery" format="NITF">wind speed</output>
          <output type="imagery" format="NITF">wind direction</output>
        </data>
      </source>
  </local>
  <web>
    <source type="web site">
      <name>COAMPS Visualization</name>
      <url>http://stratus.nrlmry.navy.mil/vis_html/tams/</url>
      <access>HTTP</access>
      <data>
        <originator type="model">
          <name>COAMPS</name>
          <alias>Coupled Ocean-Atmosphere Mesoscale Prediction System</alias>
          <location>NRL-Monterey</location>
        </originator>
        <output type="imagery" format="GIF">wind speed</output>
        <output type="imagery" format="GIF">wind direction</output>
      </data>
    </source>
  </web>
</infrastructure>
```

Figure 5. Listing of XML Infrastructure Description

```
<components>
  <java>
    <class main="true" bean="true">
      <id>1</id>
      <name>getWebGIFs</name>
      <package>LAW.Web</package>
      <bound type="String">URL</bound>
      <bound type="String">StoreDirectory</bound>
      <source type="web site">
        <access>HTTP</access>
        <data>
          <output type="imagery" format="GIF" />
        </data>
      </source>
    </class>
    <class main="true" bean="true">
      <id>2</id>
      <name>findCOAMPSGIFs</name>
      <package>LAW.COAMPS</package>
      <bound type="String">StartURL</bound>
      <bound type="String">StoreFile</bound>
      <bound type="int">ValidTime</bound>
      <source>
        <access>HTTP</access>
        <data>
          <originator type="model">
            <name>COAMPS</name>
            <alias>Coupled Ocean-Atmosphere Mesoscale Prediction System</alias>
            <location>NRL-Monterey</location>
          </originator>
          <output type="url" format="ASCII" />
        </data>
      </source>
    </class>
    <class main="true" bean="true">
      <id>3</id>
      <name>getCOAMPSData</name>
      <package>LAW.COAMPS</package>
      <bound type="String">DataSource</bound>
      <bound type="String">StoreDirectory</bound>
      <bound type="int">ValidTime</bound>
      <source type="database">
        <name>TEDS</name>
        <access>Metcast</access>
        <data>
          <originator type="model">
            <name>COAMPS</name>
            <alias>Coupled Ocean-Atmosphere Mesoscale Prediction System</alias>
            <location>NRL-Monterey</location>
          </originator>
          <output type="raw data" format="ASCII" />
          <output type="imagery" format="NITF" />
        </data>
      </source>
    </class>
  </java>
</components
```

Figure 6. Listing of XML Component Description

Assuming that the user has defined a requirement for COAMPS GIF images at a certain valid time, using either the Requirements Wizard or Agent Builder interface, the LAW must then search the metadata descriptions of the infrastructure and the existing components to determine whether it is currently possible to meet this requirement. Given the example documents above, the LAW would begin searching through the component descriptions and locate getWebGIFs with an <output> type of GIF, but with no listing of COAMPS in the <source> field. It would keep getWebGIFs in a list of components capable of providing the desired output format. It would then locate findCOAMPSGIFs with the correct <source> of COAMPS, but an <output> of ASCII URLs. The LAW would include findCOAMPSGIFs in a list of components capable of accessing the source. It would then locate getCOAMPSData, which is also capable of accessing COAMPS imagery, though in NITF format. This component would also be placed in the list of components capable of accessing the source.

Given the lists generated during this initial search, the LAW would attempt to combine the components based on their abilities to access the desired source and output the desired format. It would start with findCOAMPSGIFs at the head of the <source> list and start checking the components in the <output> list, starting with getWebGIFs. Since the output of findCOAMPSGIFs is of type "url", and the <access> type of getWebGIFs is HTTP, the LAW would be able to match these components through the grammar shown in Figure 7. This grammar tells the LAW that a <get> operation can consist of a URL object and an HTTP source, which links the output of the source component, findCOAMPSGIFs, to the input of the output component, getWebGIFs. The LAW would then place this pairing at the top of the list of possible pairings and continue searching for other possibilities. Since there are no other components capable of providing the correct output format of GIF, the LAW would continue down the list of components capable of accessing the data source. Next in the list would be getCOAMPSData, which has an output type of NITF for the imagery it retrieves from COAMPS. If there were another component capable of converting NITF to GIF, the LAW would pair it with this one and have another possible solution to provide to the user. Since there is no such translation component, the LAW has no choice but to ignore getCOAMPSData as a useless component for this requirement.

```
<get> ::= <url> from <http-source>
          | <parameter> from <sensor> at <time>
<translation> ::= <parameter> from <units> to <units>
```

Figure 7. Listing of Extended BNF Grammar Defining Component Behavior

The end result of this search through the metadata documents will be a list of components and component pairings that are capable of meeting the requirements as defined by the end-user. In the example we've just stepped through, there is a single component pairing that satisfies the requirement. The LAW then writes a skeleton agent that will hold the invocations of the Bean components (See Figure 8). The LAW calls each Bean component's get- methods to retrieve the current values of the Bean's bound properties. These properties will have been set either by hand in the Agent Builder interface or automatically through the user's answers to the Requirements Wizard (e.g., What URL would you like to retrieve from? What valid time are you interested in?). Once the LAW has values for each of the bound properties of a Bean, it writes set- statements into the agent skeleton to set the properties when the agent is invoked (See Figure 9). The LAW then writes into the agent skeleton an invocation of each Bean's run method, starting with the source Bean, followed by the output Bean.

```
//Agent created from beans:  findCOAMPSGIFs, getWebGIFs
import AgentWizard.COAMPS.*;
import AgentWizard.Web.*;
public class Agent1 {
  public static void main(String[] args) throws Exception {
  }
}
```

Figure 8. Listing of Agent Skeleton

```
//Agent created from beans:  findCOAMPSGIFs, getWebGIFs
import AgentWizard.COAMPS.*;
import AgentWizard.Web.*;
public class Agent1 {
  public static void main(String[] args) throws Exception {

      findCOAMPSGIFs source1 = new findCOAMPSGIFs();
      source1.setStartURL(
           "http://stratus.nrlmry.navy.mil/vis_html/tams/SanDiego_3/Html/");
      source1.setStoreFile("COAMPS_files.txt");
      source1.setValidTime(2001050800);
      source1.run();
      File inputFile = new File("COAMPS_files.txt");
      BufferedReader reader =
        new BufferedReader(new InputStreamReader(new
                           FileInputStream(inputFile)));
      String inputLine;
      while ((inputLine = reader.readLine()) != null)  {
        getWebGIFs output1 = new getWebGIFs();
        output1.setURL(inputLine);
        output1.setStoreDirectory("./Agent1_Output");
        output1.run();
      }
      inputFile.close();
  }
}
```

Figure 9. Listing of Complete Agent

2.8. Patent and Trademark Filing

We have submitted a preliminary patent application for the XML-based agent creation process described in the previous section. We have also submitted an application to trademark the name AgentWizard™.

2.9. Conclusion – Supporting Large-Scale Real-Time Logistics Support

With the Phase I prototype LAW, we have shown how an end-user can create agents from reusable components that perform simple operations within a limited subset of the logistics knowledge domain. If awarded the Phase II contract, this work will lead directly to the Phase I Option Task, in which we will determine a suitable multi-agent architecture on which our agents will operate. Points of concern in our choice of architecture will include capabilities for agent cooperation and collaboration, as well as the emergence of commercial and military metadata standards. The migration of our agents to a multi-agent architecture will lay the groundwork for large-scale logistics support, in which logistics personnel on local and global networks create and manipulate hundreds or thousands of agents performing autonomously on the network. During the option task, we will develop the Agent Monitor, which will provide an echelon-defined controlling end-user with the ability to stop, start, and edit agents as they operate. The internal structure of the Agent Monitor will depend heavily on the chosen agent architecture, since the Monitor will be responsible for sending and receiving messages to and from the agents using the chosen architecture's communications mechanisms.

3. References

[1] NATO Research & Technology Organization Report 8, "Land Operations in the Year 2020 (LO2020), " March 1999.

[2] Memorandum for Director of Information Systems for Command, Control, Communications and Computers (DISC4), The Pentagon, COL Larry Edmonds, CSSCS TRADOC System Manager, http://www.lee.army.mil/csscs/GCCS-A_Letter.htm.

[3] Report on Information Management to Support the Warrior, United States Air Force Scientific Advisory Board, SAB-TR-98-02, December 1998.

[4] Data Exchange Infrastructure (DEI) Public Portal, http://www.opendei.org.

[5] Control of Agent-Based Systems (COABS) Public Portal http://coabs.globalinfotek.com.

[6] DARPA Agent Markup Language (DAML) Public Portal, http://www.daml.org.

[7] ISO World, DARPA Information Systems Office web site, http://dtsn.darpa.mil/iso/index2.asp?mode=9.

[8] Advanced Logistics Project (ALP) Public Portal, http://www.darpa.mil/iso/alp.

[9] Java Beans Component Architecture Documentation, http://java.sun.com/products/javabeans/docs.

[10] Java Beans Architecture: Beans Development Kit, http://java.sun.com/products/javabeans/software/bdk_download.html.

[11] GCSS-Army DTD Documents, http://armyhti.redstone.army.mil/htdocs/dtd/dtd.htm.

[12] NTC Digital Logistic-Focused Rotation, http://gordon.army.mil/regtmktg/AC/VOL24NO4/124sig.htm.